Large-Scale Machine Learning: Randomized techniques

Jean-Philippe Vert

jean-philippe.vert@{mines-paristech,curie,ens}.fr
Outline

1. Stochastic optimization for empirical risk minimization
2. Random projections for dimension reduction
3. Random features for nonlinear embedding
4. Approximate NN
5. Shingling, hashing, sketching
## Scalability issues

<table>
<thead>
<tr>
<th>Method</th>
<th>Memory</th>
<th>Training time</th>
<th>Test time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCA</td>
<td>$O(d^2)$</td>
<td>$O(nd^2)$</td>
<td>$O(d)$</td>
</tr>
<tr>
<td>$k$-means</td>
<td>$O(nd)$</td>
<td>$O(ndk)$</td>
<td>$O(kd)$</td>
</tr>
<tr>
<td>Ridge regression</td>
<td>$O(d^2)$</td>
<td>$O(nd^2)$</td>
<td>$O(d)$</td>
</tr>
<tr>
<td>kNN</td>
<td>$O(nd)$</td>
<td>0</td>
<td>$O(nd)$</td>
</tr>
<tr>
<td>Logistic regression</td>
<td>$O(nd)$</td>
<td>$O(nd^2)$</td>
<td>$O(d)$</td>
</tr>
<tr>
<td>SVM, kernel methods</td>
<td>$O(n^2)$</td>
<td>$O(n^3)$</td>
<td>$O(nd)$</td>
</tr>
</tbody>
</table>

\[\text{O}(n^3)!\]
Today’s topic

- Trade exactness for scalability
- Compress, sketch, hash data in a smart way
- Randomization helps!

- E.g., sampling methods to approximate a mean value
Outline

1. Stochastic optimization for empirical risk minimization
2. Random projections for dimension reduction
3. Random features for nonlinear embedding
4. Approximate NN
5. Shingling, hashing, sketching
Motivation

- Classical learning theory analyzes the trade-off between:
  - approximation error (how well we approximate the true function)
  - estimation errors (how well we estimate the parameters)

- But reaching the best trade-off for a given $n$ may be impossible with limited computational resources

- We should include in the trade-off the computational budget, and see which optimization algorithm gives the best trade-off!

- Seminal paper of Bottou and Bousquet (2008)
Classical ERM setting

- Goal: learn a function $f : \mathbb{R}^d \rightarrow \mathcal{Y}$ ($\mathcal{Y} = \mathbb{R}$ or $\{-1, 1\}$)
- $P$ unknown distribution over $\mathbb{R}^d \times \mathcal{Y}$
- Training set: $S = \{(X_1, Y_1), \ldots, (X_n, Y_n)\} \subset \mathbb{R}^d \times \mathcal{Y}$ i.i.d. following $P$
- Fix a class of functions $\mathcal{F} \subset \{f : \mathbb{R}^d \rightarrow \mathbb{R}\}$
- Choose a loss $\ell(y, f(x))$
- Learning by empirical risk minimization

$$f_n \in \underset{f \in \mathcal{F}}{\arg \min} R_n[f] = \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, f(X_i))$$

- Hope that $f_n$ has a small risk:

$$R[f_n] = E\ell(Y, f_n(X))$$
Classical ERM setting

- The best possible risk is
  \[ R^* = \min_{f: \mathbb{R}^d \to \mathcal{Y}} R[f] \]

- The best achievable risk over \( \mathcal{F} \) is
  \[ R^*_\mathcal{F} = \min_{f \in \mathcal{F}} R[f] \]

- We then have the decomposition
  \[ R[f_n] - R^* = R[f_n] - R^*_\mathcal{F} + R^*_\mathcal{F} - R^* \]
  estimation error \( \epsilon_{est} \)
  approximation error \( \epsilon_{app} \)
Optimization error

- Solving the ERM problem may be hard (when \( n \) and \( d \) are large)
- Instead we usually find an approximate solution \( \tilde{f}_n \) that satisfies

\[
R_n[\tilde{f}_n] \leq R_n[f_n] + \rho
\]

- The excess risk of \( \tilde{f}_n \) is then

\[
\epsilon = R[\tilde{f}_n] - R^* = \left( R[\tilde{f}_n] - R[f_n] \right) + \epsilon_{est} + \epsilon_{app}
\]

optimization error \( \epsilon_{opt} \)
A new trade-off

\[ \epsilon = \epsilon_{app} + \epsilon_{est} + \epsilon_{opt} \]

Problem
- Choose \( \mathcal{F}, n, \rho \) to make \( \epsilon \) as small as possible
- Subject to a limit on \( n \) and on the computation time \( T \)

Table 1: Typical variations when \( \mathcal{F}, n, \) and \( \rho \) increase.

<table>
<thead>
<tr>
<th>( \mathcal{F} )</th>
<th>( n )</th>
<th>( \rho )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{E}_{app} )</td>
<td>(approximation error)</td>
<td>↘</td>
</tr>
<tr>
<td>( \mathcal{E}_{est} )</td>
<td>(estimation error)</td>
<td>↗</td>
</tr>
<tr>
<td>( \mathcal{E}_{opt} )</td>
<td>(optimization error)</td>
<td>( \cdots )</td>
</tr>
<tr>
<td>( T )</td>
<td>(computation time)</td>
<td>↗</td>
</tr>
</tbody>
</table>

Large-scale or small-scale?
- Small-scale when constraint on \( n \) is active
- Large-scale when constraint on \( T \) is active
Comparing optimization methods

$$\min_{\beta \in B \subset \mathbb{R}^d} R_n[f_\beta] = \sum_{i=1}^{n} \ell(y_i, f_\beta(x_i))$$

- Gradient descent (GD):
  $$\beta_{t+1} \leftarrow \beta_t - \eta \frac{\partial R_n(f_{\beta_t})}{\partial \beta}$$

- Second-order gradient descent (2GD), assuming Hessian $H$ known
  $$\beta_{t+1} \leftarrow \beta_t - H^{-1} \frac{\partial R_n(f_{\beta_t})}{\partial \beta}$$

- Stochastic gradient descent (SGD):
  $$\beta_{t+1} \leftarrow \beta_t - \frac{\eta}{t} \frac{\partial \ell(y_t, f_{\beta_t}(x_t))}{\partial \beta}$$
### Results (Bottou and Bousquet, 2008)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Cost of one iteration</th>
<th>Iterations to reach $\rho$</th>
<th>Time to reach accuracy $\rho$</th>
<th>Time to reach $\mathcal{E} \leq c (\mathcal{E}_{app} + \varepsilon)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>GD</td>
<td>$\mathcal{O}(nd)$</td>
<td>$\mathcal{O}(\kappa \log \frac{1}{\rho})$</td>
<td>$\mathcal{O}(n d \kappa \log \frac{1}{\rho})$</td>
<td>$\mathcal{O}\left(\frac{d^2 \kappa}{\varepsilon^{1/\alpha}} \log \frac{2}{\varepsilon} \frac{1}{\varepsilon}\right)$</td>
</tr>
<tr>
<td>2GD</td>
<td>$\mathcal{O}(d^2 + nd)$</td>
<td>$\mathcal{O}(\log \log \frac{1}{\rho})$</td>
<td>$\mathcal{O}\left((d^2 + nd) \log \log \frac{1}{\rho}\right)$</td>
<td>$\mathcal{O}\left(\frac{d^2 \kappa}{\varepsilon^{1/\alpha}} \log \frac{1}{\varepsilon} \log \log \frac{1}{\varepsilon}\right)$</td>
</tr>
<tr>
<td>SGD</td>
<td>$\mathcal{O}(d)$</td>
<td>$\frac{\nu \kappa^2}{\rho} + o\left(\frac{1}{\rho}\right)$</td>
<td>$\mathcal{O}\left(\frac{d \nu \kappa^2}{\rho}\right)$</td>
<td>$\mathcal{O}\left(\frac{d \nu \kappa^2}{\varepsilon}\right)$</td>
</tr>
</tbody>
</table>

- $\alpha \in [1/2, 1]$ comes from the bound on $\varepsilon_{est}$ and depends on the data
- In the last column, $n$ and $\rho$ are optimized to reach $\varepsilon$ for each method
- 2GD optimizes much faster than GD, but limited gain on the final performance limited by $\varepsilon^{-1/\alpha}$ coming from the estimation error
- SGD:
  - Optimization speed is catastrophic
  - Learning speed is the best, and independent of $\alpha$
- This suggests that **SGD is very competitive** (and has become the de facto standard in large-scale ML)
Illustration

**Results: Linear SVM**

\[ \ell(\hat{y}, y) = \max\{0, 1 - y\hat{y}\} \quad \lambda = 0.0001 \]

<table>
<thead>
<tr>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVMLight</td>
<td>23,642 secs</td>
<td>0.2275</td>
</tr>
<tr>
<td>SVMPerf</td>
<td>66 secs</td>
<td>0.2278</td>
</tr>
<tr>
<td>SGD</td>
<td>1.4 secs</td>
<td>0.2275</td>
</tr>
</tbody>
</table>

**Results: Log-Loss Classifier**

\[ \ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y})) \quad \lambda = 0.00001 \]

<table>
<thead>
<tr>
<th>Training Time</th>
<th>Primal cost</th>
<th>Test Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRON(LibLinear, (\varepsilon = 0.01))</td>
<td>30 secs</td>
<td>0.18907</td>
</tr>
<tr>
<td>TRON(LibLinear, (\varepsilon = 0.001))</td>
<td>44 secs</td>
<td>0.18890</td>
</tr>
<tr>
<td>SGD</td>
<td>2.3 secs</td>
<td>0.18893</td>
</tr>
</tbody>
</table>

Outline

1. Stochastic optimization for empirical risk minimization

2. Random projections for dimension reduction

3. Random features for nonlinear embedding

4. Approximate NN

5. Shingling, hashing, sketching
Issues when $d$ is large

- Affects scalability of algorithms, e.g., $O(nd)$ for kNN or $O(d^3)$ for ridge regression
- Hard to visualize
- (Sometimes) counterintuitive phenomena in high dimension, e.g., concentration of measure for Gaussian data

- Statistical inference degrades when $d$ increases (curse of dimension)
Dimension reduction with PCA

- Projects data onto $k < d$ dimensions that captures the largest amount of variance.
- Also minimizes total reconstruction errors:
  \[ \min_{S_k} \sum_{i=1}^{n} \| x_i - \Pi_{S_k}(x_i) \|^2 \]
- But computational expensive: $O(nd^2)$
- No theoretical guarantee on distance preservation.
Linear dimension reduction

\[ X' = X \times R \]

Can we find \( R \) efficiently?
Can we preserve distances?

\[ \forall i, j = 1, \ldots, n, \quad \| f(x_i) - f(x_j) \| \approx \| x_i - x_j \| \]

Note: when \( d > n \), we can take \( k = n \) and preserve all distances exactly (kernel trick)
Random projections

Simply take a random projection matrix:

\[ f(x) = \frac{1}{\sqrt{k}} R^\top x \quad \text{with} \quad R_{ij} \sim \mathcal{N}(0, 1) \]

Theorem (Johnson and Lindenstrauss, 1984)

For any \( \epsilon > 0 \) and \( n \in \mathbb{N} \), take

\[ k \geq 4 \left( \frac{\epsilon^2}{2} - \frac{\epsilon^3}{3} \right)^{-1} \log(n) \approx \epsilon^{-2} \log(n) \]

Then the following holds with probability at least \( 1 - \frac{1}{n} \):

\[ \forall i, j = 1, \ldots, n \quad (1 - \epsilon) \| x_i - x_j \|^2 \leq \| f(x_i) - f(x_j) \|^2 \leq (1 + \epsilon) \| x_i - x_j \|^2 \]

- \( k \) does not depend on \( d \)!
- \( n = 1M, \epsilon = 0.1 \implies k \approx 5K \)
- \( n = 1B, \epsilon = 0.1 \implies k \approx 8K \)
Proof (1/3)

- For a single dimension, $q_j = r_j^T u$:

$$E(q_j) = E(r_j)^T u = 0$$
$$E(q_j)^2 = u^T E(r_j r_j^T) u = \| u \|^2$$

- For the $k$-dimensional projection $f(u) = 1/\sqrt{kR^T u}$:

$$\| f(u) \|^2 = \frac{1}{k} \sum_{j=1}^{k} q_j^2 \sim \frac{\| u \|^2}{k} \chi^2(k)$$

$$E\| f(u) \|^2 = \frac{1}{k} \sum_{j=1}^{k} E(q_j^2) = \| u \|^2$$

- Need to show that $\| f(u) \|^2$ is concentrated around its mean
Proof (2/3)

\[ P \left[ \| f(u) \|^2 > (1 + \epsilon)\| u \|^2 \right] \\
= P \left[ \chi^2(k) > (1 + \epsilon)k \right] \\
= P \left[ e^{\lambda \chi^2(k)} > e^{\lambda(1+\epsilon)k} \right] \\
\leq E \left[ e^{\lambda \chi^2(k)} \right] e^{-\lambda(1+\epsilon)k} \\
= (1 - 2\lambda)^{-\frac{k}{2}} e^{-\lambda(1+\epsilon)k} \\
= \left((1 + \epsilon)e^{-\epsilon}\right)^{k/2} \\
\leq e^{-\left(\epsilon^2/2 - \epsilon^3/3\right)k/2} \\
= n^{-2} \\
\]

(for any \( \lambda > 0 \))

(Markov)

(MGF of \( \chi^2(k) \) for \( 0 \leq \lambda \leq 1/2 \))

(take \( \lambda = \epsilon/2(1 + \epsilon) \))

(use \( \log(1 + x) \leq x - x^2/2 + x^3/3 \))

(take \( k = 4 \left(\epsilon^2/2 - \epsilon^3/3\right) \log(n) \))

Similarly we get

\[ P \left[ \| f \|^2 < (1 - \epsilon)\| u \|^2 \right] < n^{-2} \]
Proof (3/3)

- Apply with $u = x_i - x_j$ and use linearity of $f$ to show that for an $(x_i, x_j)$ pair, the probability of large distortion is $\leq 2n^{-2}$.

- Union bound: for all $n(n-1)/2$ pairs, the probability that at least one has large distortion is smaller than

$$\frac{n(n-1)}{2} \times \frac{2}{n^2} = 1 - \frac{1}{n}$$
Scalability

- $n = O(1B); d = O(1M) \implies k = O(10K)$
- Memory: need to store $R$, $O(dk) \approx 40\text{GB}$
- Computation: $X \times R$ in $O(ndk)$
- Other random matrices $R$ have similar properties but better scalability, e.g.:
  - "add or subtract" (Achlioptas, 2003), 1 bit/entry, size $\approx 1,25\text{GB}$
    \[
    R_{ij} = \begin{cases} 
    +1 & \text{with probability } 1/2 \\
    -1 & \text{with probability } 1/2 
    \end{cases}
    \]
  - Fast Johnson-Lindenstrauss transform (Ailon and Chazelle, 2009)
    where $R = PHD$, compute $f(x)$ in $O(d \log d)$

\[
\begin{pmatrix}
\text{Sparse} \\
\text{JL}
\end{pmatrix}
\begin{pmatrix}
\text{Walsh–} \\
\text{Hadamard}
\end{pmatrix}
\begin{pmatrix}
\pm 1 \\
\vdots \\
\pm 1
\end{pmatrix}
\]
Outline

1. Stochastic optimization for empirical risk minimization
2. Random projections for dimension reduction
3. Random features for nonlinear embedding
4. Approximate NN
5. Shingling, hashing, sketching
Motivation

\[ \mathbb{R}^d \rightarrow \mathbb{R}^D \rightarrow \mathbb{R}^k \]

Kernel Phi

JL random projection

Random features?
Fourier feature space

Example: Gaussian kernel

\[
e^{-\frac{\|x-x'\|^2}{2}} = \frac{1}{(2\pi)^{d/2}} \int_{\mathbb{R}^d} e^{i\omega^\top (x-x')} e^{-\frac{\|\omega\|^2}{2}} d\omega
\]

\[= E_\omega \cos \left( \omega^\top (x-x') \right)\]

\[= E_{\omega,b} \left[ 2 \cos \left( \omega^\top x + b \right) \cos \left( \omega^\top x' + b \right) \right]\]

with

\[
\omega \sim p(d\omega) = \frac{1}{(2\pi)^{d/2}} e^{-\frac{\|\omega\|^2}{2}} d\omega, \quad b \sim \mathcal{U}([0, 2\pi]).
\]

This is of the form \(K(x, x') = \Phi(x)^\top \Phi(x')\) with \(D = +\infty:\)

\[
\Phi : \mathbb{R}^d \to L_2 \left( \left( \mathbb{R}^d, p(d\omega) \right) \times ([0, 2\pi], \mathcal{U}) \right)
\]
Random Fourier features (Rahimi and Recht, 2008)

- For \( i = 1, \ldots, k \), sample randomly:

\[
(\omega_i, b_i) \sim p(d\omega) \times U([0, 2\pi])
\]

- Create random features:

\[
\forall x \in \mathbb{R}^d, \quad f_i(x) = \sqrt{\frac{2}{k}} \cos(\omega_i^T x + b_i)
\]
Random Fourier features (Rahimi and Recht, 2008)

For any $x, x' \in \mathbb{R}^d$, it holds

$$E \left[ f(x)^\top f(x') \right] = \sum_{i=1}^{k} E \left[ f_i(x)f_i(x') \right]$$

$$= \frac{1}{k} \sum_{i=1}^{k} E \left[ 2 \cos (\omega^\top x + b) \cos (\omega^\top x' + b) \right]$$

$$= K(x, x')$$

and by Hoeffding's inequality,

$$P \left[ \left| f(x)^\top f(x') - K(x, x') \right| > \epsilon \right] \leq 2e^{-\frac{k\epsilon^2}{2}}$$

This allows to approximate learning with the Gaussian kernel with a simple linear model in $k$ dimensions!
Generalization

A translation-invariant (t.i.) kernel is of the form

$$K(x, x') = \varphi(x - x')$$

Bochner’s theorem

For a continuous function $\varphi : \mathbb{R}^d \to \mathbb{R}$, $K$ is p.d. if and only if $\varphi$ is the Fourier-Stieltjes transform of a symmetric and positive finite Borel measure $\mu \in M(\mathbb{R}^d)$:

$$\varphi(x) = \int_{\mathbb{R}^d} e^{-i\omega^\top x} d\mu(\omega)$$

Just sample $\omega_i \sim \frac{d\mu(\omega)}{\mu(\mathbb{R}^d)}$ and $b_i \sim \mathcal{U}([0, 2\pi])$ to approximate any t.i. kernel $K$ with random features

$$\sqrt{\frac{2}{k}} \cos \left( \omega_i^\top x + b_i \right)$$
Examples

\[ K(x, x') = \varphi(x - x') = \int_{\mathbb{R}^d} e^{-i\omega^\top(x-x')} d\mu(\omega) \]

<table>
<thead>
<tr>
<th>Kernel</th>
<th>( \varphi(x) )</th>
<th>( \mu(d\omega) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian</td>
<td>( \exp\left(-\frac{|x|^2}{2}\right) )</td>
<td>((2\pi)^{-d/2} \exp\left(-\frac{|\omega|^2}{2}\right) )</td>
</tr>
<tr>
<td>Laplace</td>
<td>( \exp\left(-|x|_1\right) )</td>
<td>( \prod_{i=1}^{k} \frac{1}{\pi\left(1+\omega_i^2\right)} )</td>
</tr>
<tr>
<td>Cauchy</td>
<td>( \prod_{i=1}^{k} \frac{2}{1+x_i^2} )</td>
<td>( e^{-|\omega|_1} )</td>
</tr>
</tbody>
</table>
### Performance (Rahimi and Recht, 2008)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fourier+LS</th>
<th>Binning+LS</th>
<th>CVM</th>
<th>Exact SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>3.6%</td>
<td>5.3%</td>
<td>5.5%</td>
<td>11%</td>
</tr>
<tr>
<td>regression 6500 instances 21 dims</td>
<td>20 secs</td>
<td>3 mins</td>
<td>51 secs</td>
<td>31 secs</td>
</tr>
<tr>
<td></td>
<td>$D = 300$</td>
<td>$P = 350$</td>
<td>ASVM</td>
<td>ASVM</td>
</tr>
<tr>
<td>Census</td>
<td>5%</td>
<td>7.5%</td>
<td>8.8%</td>
<td>9%</td>
</tr>
<tr>
<td>regression 18,000 instances 119 dims</td>
<td>36 secs</td>
<td>19 mins</td>
<td>7.5 mins</td>
<td>13 mins</td>
</tr>
<tr>
<td></td>
<td>$D = 500$</td>
<td>$P = 30$</td>
<td>SVMTorch</td>
<td>SVMTorch</td>
</tr>
<tr>
<td>Adult</td>
<td>14.9%</td>
<td>15.3%</td>
<td>14.8%</td>
<td>15.1%</td>
</tr>
<tr>
<td>classification 32,000 instances 123 dims</td>
<td>9 secs</td>
<td>1.5 mins</td>
<td>73 mins</td>
<td>7 mins</td>
</tr>
<tr>
<td></td>
<td>$D = 500$</td>
<td>$P = 30$</td>
<td>SVM^light</td>
<td>SVM^light</td>
</tr>
<tr>
<td>Forest Cover</td>
<td>11.6%</td>
<td>2.2%</td>
<td>2.3%</td>
<td>2.2%</td>
</tr>
<tr>
<td>classification 522,000 instances 54 dims</td>
<td>71 mins</td>
<td>25 mins</td>
<td>7.5 hrs</td>
<td>44 hrs</td>
</tr>
<tr>
<td></td>
<td>$D = 5000$</td>
<td>$P = 50$</td>
<td>libSVM</td>
<td>libSVM</td>
</tr>
<tr>
<td>KDDCUP99 (see footnote) classification</td>
<td>7.3%</td>
<td>7.3%</td>
<td>6.2% (18%)</td>
<td>8.3%</td>
</tr>
<tr>
<td>4,900,000 instances 127 dims</td>
<td>1.5 min</td>
<td>35 mins</td>
<td>1.4 secs (20 secs)</td>
<td>&lt; 1 s</td>
</tr>
<tr>
<td></td>
<td>$D = 50$</td>
<td>$P = 10$</td>
<td>SVM+sampling</td>
<td>SVM+sampling</td>
</tr>
</tbody>
</table>
Outline

1. Stochastic optimization for empirical risk minimization
2. Random projections for dimension reduction
3. Random features for nonlinear embedding
4. Approximate NN
5. Shingling, hashing, sketching
Motivation

- Database $\mathcal{S} = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, query $q \in \mathbb{R}^d$
- Naively: $O(nd)$ to compute distances $\|q - x_i\|$ and find the smallest one
- For $n = 1B$, $d = 10k$, it takes 15 hours
- Projections $\mathbb{R}^d \rightarrow \mathbb{R}^k$ with $k < d$ is not good enough if $n$ is large
Given $\epsilon > 0$, the approximate nearest neighbor (ANN) problem is:

Find $y \in S$ such that $\| q - y \| \leq (1 + \epsilon) \min_{x \in S} \| q - x \|$

Two popular ANN approaches

1. Tree approaches
   - Recursively partition the data: Divide and Conquer
   - Expected query time: $O(\log(n))$
   - Many variants: KDtree, Balltree, PCA-tree, Vantage Point tree
   - Shown to perform very well in relatively low-dim data

2. Hashing approaches
   - Each image in database represented as a code
   - Significant reduction in storage
   - Expected query time: $O(1)$ or $O(n)$
   - Compact codes preferred
**KD tree**

- **Axis-parallel splits**
- Along the direction of largest variance
- Split along the median $\rightarrow$ balanced partitioning
- Split recursively until each node has a single data point
Search in a KD tree

- Finds the leaf of the query in $O(\log(n))$
- But backtracking is needed to visit other leaves surrounding the cell
- As $d$ increases, the number of leaves to visit grows exponentially
- Complexity: $O(nd \log(n))$ to build the tree, $O(nd)$ to store the original data
- Works fine up to $d = 10 \sim 100$
Variants

Vantage Point (VP)-Tree

Building VP-Tree
– Select a vantage point randomly (usually from data periphery)
– Compute distance from all other points and pick median distance
– Binary tree: split data using median distance from vantage point
– Split recursively until each node has only one point (leaves)

VP-Tree vs KD-Tree

Sanjiv Kumar
10/5/2010 EECS6898 – Large Scale Machine Learning

VP-Tree

left right
median distance

VP-Tree

Yianilos [3]
Variants

Ball tree

- Find the (approx) diameter of the given dataset
- Find the point farthest from mean and another farthest from it
- Threshold at median
- Another variation: split according to distance from two points (i.e., threshold at mid point of line joining two centers), need to store two vectors per node

Susceptible to outliers!

Sanjiv Kumar 10/5/2010 EECS6898 – Large Scale Machine Learning

PCA tree

- Use top eigenvector of data covariance as projection direction
- Threshold at median
- More robust than ball tree in presence of outliers

Expensive, not enough data at lower levels to construct covariance!

Sanjiv Kumar 10/5/2010 EECS6898 – Large Scale Machine Learning

Random-Projection (RP) Tree

- Randomly sample a projection direction from a fixed distribution
- Threshold at “adjusted” median
- Robust for high-dim data
- Can adapt to low-dimensional structure in the data well
Binary code using multiple hashing

No recursive partitioning, unlike trees!

ANN with codes:
1. Choose a set of binary hashing functions to design a binary code
2. Index the database = compute codes for all points
3. Querying: compute the code of the query, and retrieve the points with similar codes
Hashing

A hash function is a function $h : \mathcal{X} \rightarrow \mathcal{Z}$ where

- $\mathcal{X}$ is the set of data ($\mathbb{R}^d$ for us)
- $\mathcal{Z} = \{1, \ldots, N\}$ is a finite set of codes

There is a collision when $h(x) = h(x')$ for two different entries $x \neq x'$

https://en.wikipedia.org/wiki/Hash_function
Locality sensitive hashing (LSH)

- Let a random hash function \( h : \mathcal{X} \rightarrow \mathcal{Z} \)
- It is a LSH with respect to a similarity function \( sim(x, x') \) on \( \mathcal{X} \) if there exists a monotonically increasing function \( f : \mathbb{R} \rightarrow [0, 1] \) such that:
  \[
  \forall x, x' \in \mathcal{X}, \quad P[h(x) = h(x')] = f(sim(x, x'))
  \]
- "Probability of collision increases with similarity"

![Diagram of LSH](image)
Example: *simHash*

\[ r^T x > 0 \quad r^T x < 0 \]

\[ h_r(x) = \begin{cases} 
1 & \text{if } r^T x \geq 0 \\
0 & \text{otherwise.}
\end{cases} \]

\[ P \left[ h_r(x) = h_r(x') \right] = 1 - \frac{\theta}{\pi} \]

LSH with respect to the cosine similarity \( \text{sim}(x, x') = \cos(\theta) \) (Goemans and Williamson, 1995).
ANN with LSH

Given:

\[ \Pr(h(x) = h(y)) = f(\text{sim}(x, y)), \quad f \text{ is monotonic.} \]

\( h_1 \), \( h_2 \) Buckets (pointers only)

<table>
<thead>
<tr>
<th>( h_1 )</th>
<th>( h_2 )</th>
<th>Buckets (pointers only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>...</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>...</td>
</tr>
<tr>
<td>00</td>
<td>10</td>
<td>Empty</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>...</td>
</tr>
</tbody>
</table>

- \( h_i(q) = h_i(x) \) implies high similarity (locality sensitive)

\[ h_1, h_2 : \mathbb{R}^D \rightarrow \{0, 1, 2, 3\} \]
ANN with LSH

Table 1

<table>
<thead>
<tr>
<th>$h^1_1$</th>
<th>...</th>
<th>$h^1_K$</th>
<th>Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 ...</td>
<td>00</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>00 ...</td>
<td>01</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>00 ...</td>
<td>10</td>
<td>Empty</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11 ...</td>
<td>11</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Table L

<table>
<thead>
<tr>
<th>$h^L_1$</th>
<th>...</th>
<th>$h^L_K$</th>
<th>Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 ...</td>
<td>00</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>00 ...</td>
<td>01</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>00 ...</td>
<td>10</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>11 ...</td>
<td>11</td>
<td>Empty</td>
<td></td>
</tr>
</tbody>
</table>

- $h_i(q) = h_i(x)$ implies high similarity (locality sensitive)
- Use $K$ contentions, repeated in $L$ tables
- Querying: report union of $L$ buckets
- Choice of $K$ and $L$:
  - Large $K$ increases precision but decreases recall
  - Large $L$ increases recall but also storage
  - Optimization is possible to minimize run-time for a given application
Choice of $K$ and $L$

<table>
<thead>
<tr>
<th>$h^1_1$</th>
<th>...</th>
<th>$h^1_K$</th>
<th>Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>...</td>
<td>00</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>00</td>
<td>...</td>
<td>01</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>00</td>
<td>...</td>
<td>10</td>
<td>Empty</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>...</td>
<td>11</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$h^L_1$</th>
<th>...</th>
<th>$h^L_K$</th>
<th>Buckets</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>...</td>
<td>00</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>00</td>
<td>...</td>
<td>01</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>00</td>
<td>...</td>
<td>10</td>
<td>[Diagram]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>...</td>
<td>11</td>
<td>Empty</td>
</tr>
</tbody>
</table>

- Large $K$ increases precision but decreases recall
- Large $L$ increases recall but also storage

If $P(h(x) = h(q)) = S$, then

$$P(x \text{ is among the candidate NN}) = 1 - \left(1 - S^K\right)^L$$
S-curve

\[ 1 - (1 - S^r)^b \]
LSH for $\| x - x' \|_s$?

$$h_k(x) = \left\lfloor \frac{w_k^\top x + b_k}{t} \right\rfloor \quad w_k \sim \prod_{i=1}^{d} P_s(w_k^i), \quad b_k \sim U([0, t])$$

- $P_s$ a $s$-stable distribution, i.e., for any $x \in \mathbb{R}^d$, and any $w$ i.i.d. with $w^i \sim P_s$, $x^\top w \sim \| x \|_s w^1$.
- $s$-stable distributions exist for $p \in (0, 2]$:
  - Gaussian $\mathcal{N}(0, 1)$ is 2-stable
  - Cauchy $dx / (\pi (1 + x^2))$ is 1-stable
- Then $P[h_k(x) = h_k(x')]$ increases as $\| x - x' \|_s$ decreases
Outline

1. Stochastic optimization for empirical risk minimization
2. Random projections for dimension reduction
3. Random features for nonlinear embedding
4. Approximate NN
5. Shingling, hashing, sketching
Motivation

- The hashing / LSH trick is a fast random projection to compact binary codes.
- Initially proposed for ANN problems, it can also be used for more general learning problems.
- It is particularly effective when data are first converted to huge binary vectors, using a specific similarity measure (the resemblance).
- Applications: texts, time series, images...
Shingling and resemblance

- Given some input space $\mathcal{X}$ (e.g., texts, times series...), a shingling is a representation as a large binary vector

$$x \in \{0, 1\}^D$$

- Equivalently, represent $x$ as a subset of $S_x \subset \Omega = \{0, \ldots, D - 1\}$

- Example: represent a text by the set of $w$-shingles it contains, i.e., sequences of $w$ words. Typically, $w = 5 \times 10^5$ words, $D = 10^2 5$, but very sparse.

- A common measure of similarity between two such vectors is the resemblance (a.k.a. Jaccard or Tanimoto similarity):

$$R(x_1, x_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

- But computing $R(x_1, x_2)$ is expensive, and not scalable for NN search or machine learning
Minwise hashing

- Let $\pi \in \mathcal{S}_D$ be a random permutation of $\Omega$
- Let $h_\pi : \{0, 1\}^D \to \Omega$ assign to $S \subset \Omega$ the smallest index of $\pi(S)$:
  \[ h_\pi(x) = \min \{ \pi(i) : i \in S_x \} \]

**Theorem (Broder, 1997)**

Minwise hashing is a LSH with respect to the resemblance:

\[ P[h_\pi(x_1) = h_\pi(x_2)] = R(x_1, x_2) \]

**Proof:**

- The smallest index $\min(h_\pi(x_1), h_\pi(x_2))$ correspond a random element of $S_1 \cup S_2$
- $h_\pi(x_1) = h_\pi(x_2)$ if it is in $S_1 \cap S_2$
- This happens with probability $R(x_1, x_2)$
Applications of minwise hashing

- If we pick $k$ random permutations, we can represent $x$ by $(h_1(x), \ldots, h_k(x)) \in \{0, 1\}^{Dk}$
- Used for ANN, using the general LSH technique discussed earlier
- Learning linear models as an approximation to learning a nonlinear function with the resemblance kernel
- Various tricks to improve scalability
  - $b$-bit minwise hashing (Li and König, 2010): only keep the last $b$ bits of $h_\pi(x)$, which reduces the dimensionality of the hashed matrix to $2^b k$
  - One-permutation hashing (Li et al., 2012): use a single permutation, keep the smallest index in each consecutive block of size $k$

As illustrated in Figure 1, the idea of minwise hashing can be potentially used for training linear SVM and logistic regression on high-dimensional binary data by converting the permuted data into a new data matrix in reduced dimensions. This of course would not be realistic if the total number of nonzeros for some set is small compared to $k$, and we will present strategies on how to deal with empty bins should they occur.

Figure 1:

We will show that empty bins occur rarely unless the total number of nonzeros for some set is small compared to $k$, and we will present strategies on how to deal with empty bins should they occur.

As illustrated in Figure 1, the idea of minwise hashing can be potentially used for training linear SVM and logistic regression on high-dimensional binary data by converting the permuted data into a new data matrix in reduced dimensions. This of course would not be realistic if the total number of nonzeros for some set is small compared to $k$, and we will present strategies on how to deal with empty bins should they occur.

\[ \pi(S_1) = 0 0 1 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 \]
\[ \pi(S_2) = 1 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 \]
\[ \pi(S_3) = 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 \]

This shows in particular that the resemblance is positive definite.
Goal: improve the scalability of random projections or minwise hashing, both in memory (sparsity) and processing time

Simple idea:

- Let \( h : [1, d] \to [1, k] \) a hash function
- For \( x \in \mathbb{R}^d \) (or \( \{0, 1\}^d \)) let \( \Phi(x) \in \mathbb{R}^k \) with

\[
\forall i = 1, \ldots, k \quad \Phi_i(x) = \sum_{j \in [1,d]: h(j)=i} x_j
\]

"Accumulate coordinates \( i \) of \( x \) for which \( h(i) \) is the same
- Repeat \( L \) times and concatenate if needed, to limit the effect of collisions

Advantages

- No memory needed for projections (vs. LSH)
- No need for dictionary (just a hash function that can hash anything)
- Sparsity preserving
Outline

1. Stochastic optimization for empirical risk minimization
2. Random projections for dimension reduction
3. Random features for nonlinear embedding
4. Approximate NN
5. Shingling, hashing, sketching
Conclusion

- Randomization is a powerful idea to trade exactness for scalability
- Often in ML, we do not care about exactness, only about a sufficiently accurate solution
- Theoretical guarantees in high probability (only)


